



**APPLICATION  
FOR  
UNITED STATES LETTERS PATENT**

**TITLE:** SYNCHRONIZATION OF RECURRING RECORDS IN  
INCOMPATIBLE DATABASES

**APPLICANT:** DAVID J. BOOTHBY

08752490-11396

"EXPRESS MAIL" Mailing Label Number EM121269025US

Date of Deposit November 13, 1996

I hereby certify under 37 CFR 1.10 that this correspondence is being deposited with the United States Postal Service as "Express Mail Post Office To Addressee" with sufficient postage on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231.

Lisa G. Gray  
Lisa G. Gray



**DRAWINGS**  
**FOR**  
**APPLICATION**  
**FOR**  
**UNITED STATES LETTERS PATENT**

**TITLE:**                   **SYNCHRONIZATION OF RECURRING RECORDS IN  
INCOMPATIBLE DATABASES**

**APPLICANT:**           **DAVID J. BOOTHBY**

"EXPRESS MAIL" Mailing Label Number EM121269025US

Date of Deposit November 13, 1996  
I hereby certify under 37 CFR 1.10 that this correspondence  
is being deposited with the United States Postal Service as  
"Express Mail Post Office To Addressee" with sufficient  
postage on the date indicated above and is addressed to the  
Assistant Commissioner for Patents, Washington, D.C.  
20231.

USA G Gray  
USA G Gray



PATENT  
ATTORNEY DOCKET NO: 05110/006001

SYNCHRONIZATION OF RECURRING  
RECORDS IN INCOMPATIBLE DATABASES

5 Reference to Microfiche Appendix

An appendix (appearing now in paper format to be replaced later in microfiche format) forms part of this application. The appendix, which includes a source code listing relating to an embodiment of the invention, includes

10 <sup>691 frames on 8 microfiche</sup>  
~~frames on microfiche.~~

This patent document (including the microfiche appendix) contains material that is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document as it appears in the Patent and Trademark Office file or records, but otherwise reserves all copyright rights whatsoever.

Background of the Invention

20 This invention relates to synchronizing incompatible databases.

Databases are collections of data entries which are organized, stored, and manipulated in a manner specified by applications known as database managers (hereinafter also referred to as "Applications"). The manner in which database entries are organized in a database is known as the data structure. There are generally two types of database managers. First are general purpose database managers in which the user determines (usually at the outset, but subject to future revisions) what the data structure is.

30 These Applications often have their own programming language and provide great flexibility to the user. Second are special purpose database managers that are specifically designed to create and manage a database having a preset

08752490-11396

data structure. Examples of these special purpose database managers are various scheduling, diary, and contact manager Applications for desktop and handheld computers. Database managers organize the information in a database into  
5 records, with each record made up of fields. Fields and records of a database may have many different characteristics depending on the database manager's purpose and utility.

Databases can be said to be incompatible with one  
10 another when the data structure of one is not the same as the data structure of another, even though some of the content of the records is substantially the same. For example, one database may store names and addresses in the following fields: FIRST\_NAME, LAST\_NAME, and ADDRESS.  
15 Another database may, however, store the same information with the following structure: NAME, STREET\_NO., STREET\_NAME, CITY\_STATE, and ZIP. Although the content of the records is intended to contain the same kind of information, the organization of that information is  
20 completely different.

It is often the case that users of incompatible databases want to be able to synchronize the databases. For example, in the context of scheduling and contact manager Applications, a person might use one Application on the  
25 desktop computer at work <sup>and</sup> ~~while~~ another on his handheld computer or his laptop computer at home. It is desirable for many of these users to be able to synchronize the entries on one with entries on another. However, the incompatibility of the two databases creates many problems  
30 that need to be solved for successful synchronization. The U.S. patent and copending patent application of the assignee hereof, IntelliLink Corp., of Nashua, New Hampshire (U.S. Patent No. 5,392,390; U.S. Application, Serial No.

*B*  
08/371,194, filed on January 11, 1995, <sup>now US Patent No. 5,684,990,</sup> incorporated by reference herein) show two methods for synchronizing incompatible databases and solving some of the problems arising from incompatibility of databases. However, other  
5 problems remain.

One kind of incompatibility is when one database manager uses recurring records. Recurring records are single records which contain information which indicates that the records actually represent multiple records sharing  
10 some common information. Many scheduling Applications, for example, permit as a single record an event which occurs regularly over a period of time. Instances of such entries are biweekly committee meetings or weekly staff lunches. Other scheduling Applications do not use these types of  
15 records. A user has to create equivalent entries by creating a separate record for each instance of these recurring events.

Various problems arise when synchronizing these types of records. Let us consider a situation when  
20 Application A uses recurring records while Application B does not. A synchronizing application must be able to create multiple entries for B for each recurring entry in A. It also must be able to identify some of the records in database B as instances of recurring records in database A.  
25 Also, many Applications which allow recurring records also permit revision and editing of single instances of recurring records without affecting the master recurring record. Moreover, single instances of a recurring event in Application B may be changed or deleted. The recurring  
30 master may also be changed which has the effect of changing all instances. These changes make it harder to identify multiple entries in database B as instances of a recurring record in database A. Moreover, synchronization must take

these changes into account when updating records in one or the other database.

### Summary of the Invention

5 The invention provides a technique for synchronizing  
databases in which different techniques are used for storing  
a recurring event. A database in which the recurring event  
is, for example, stored as a single recurring record can be  
synchronized with a database in which the same recurring  
event is stored as a series of individual records. The  
10 individual records are processed to form a synthetic  
recurring record representing the set of individual records,  
and synchronization decisions are based on a comparison of  
the synthetic record to the recurring record of the other  
database. Following synchronization, the synthetic record  
15 can be "fanned" back into the individual records to update  
the database containing individual records, and the updated  
recurring record can be written back to the other database.  
In this way, the invention avoids the problems encountered  
with prior methods, in which synchronization resulted in a  
20 recurring record being transformed into a series of  
individual records.

5 The invention features a computer implemented method  
of synchronizing at least a first and a second database,  
wherein the manner of storing a set of recurring instances  
differs between the first and second databases, and at least  
25 the first database uses a recurring record to store the set  
of recurring instances. A plurality of instances in the  
second database are processed to generate a synthetic  
recurring record representing recurring instances in the  
second database, the synthetic recurring record of the  
30 second database is compared to a recurring record of the

first database, and synchronization is completed based on the outcome of the comparison.

Preferred embodiments of the invention may include one or more of the following features: Completing

5 synchronization may include adding, modifying, or deleting the synthetic recurring record or the recurring record. Following synchronization, the synthetic recurring record may be fanned back into a plurality of single instances. The set of recurring instances may be stored in the second  
10 database as a plurality of single instances. The set of recurring instances may be stored in the second database as a recurring record having a different record structure than the recurring record of the first database. A history file may be stored containing a record representative of the  
15 presence of a recurring record or a synthetic recurring record in past synchronizations.

The invention may be implemented in hardware or software, or a combination of both. Preferably, the technique is implemented in computer programs executing on  
20 programmable computers that each include a processor, a storage medium readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. Program code is applied to data entered using the input device to  
25 perform the functions described above and to generate output information. The output information is applied to one or more output devices.

Each program is preferably implemented in a high level procedural or object oriented programming language to  
30 communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or interpreted language.

Each such computer program is preferably stored on a storage medium or device (e.g., ROM or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform the procedures described in this document. The system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured causes a computer to operate in a specific and predefined manner.

Other features and advantages of the invention will become apparent from the following description of preferred embodiments, including the drawings, and from the claims.

#### Brief Description of the Drawing

Figure 1 is a schematic drawing of the various modules constituting the preferred embodiment.

Figure 2 is a representation of the Workspace data array.

Figure 3 is the pseudocode for the Translation Engine Control Module.

Figure 4 is the pseudocode for generating the parameter Table.

Figure 5 is the pseudocode for fanning a recurring record.

Figure 6 is the pseudocode for the Synchronizer loading the History File.

Figure 7 is the pseudocode for matching key fields (Key\_Field\_Match).

Figure 8 is the pseudocode for loading records of B\_Database into Workspace.

03752490-111396

C1  
C2

7



Figure 9 is the pseudocode for A\_Sanitization of B\_Database records in Workspace.

Figure 10 is the Pseudocode for a specific example of a rule of data value used for sanitization.

5 Figure 11 is the pseudocode for orientation analysis.

Figure 12 is the pseudocode for Conflict Analysis And Resolution (CAAR).

10 Figure 13 is the pseudocode for analyzing unique ID bearing Fanned Instance Groups (FIGs).

Figure 14 is the pseudocode for expanding CIGs created from unique ID bearing records.

Figure 15 is the pseudocode for finding weak matches for a record.

15 Figure 16 is the pseudocode for finding matches between recurring items and non\_unique ID bearing instances.

Figure 17 is the pseudocode for completing Same Key Group (SKG) analysis.

20 Figure 18 is the pseudocode for setting the Maximum\_CIG\_Size for every CIG analyzed in Figure 17.

Figure 19 is the pseudocode for setting CIG\_Types.

Figure 20 is the User Interface for conflict resolution when the Notify option is selected.

25 Figure 21 is the pseudocode for merging exclusion lists.

Figure 22 is a look up table used by the function in Fig. 21.

Figure 23 is a look up table used by the function in Fig. 21.

30 Figure 24 is a look up table used by the function in Fig. 21.

Figure 25 is a pseudocode for unloading records from Workspace to a non-rebuild-all database.

Figure 26 is the look up table for determining unloading outcome results.

Figure 27 is the pseudocode for fanning recurring records of A-Database for unloading.

Figure 28 is the pseudocode for unloading the History File.

Figure 29 is a table showing cases in which Recurring Masters are fanned into own database.

Figure 30 is the pseudocode for loading records by a fast synchronization Translator.

Figure 31 is the pseudocode for loading records by a fast synchronization Translator.

#### Description of the Preferred Embodiments

Fig. 1 shows the relationship between the various modules of the preferred embodiment. Translation Engine 1 comprises Control Module 2 and Parameters Table Generator 3. Control Module 2 is responsible for controlling the synchronizing process by instructing various modules to perform specific tasks on the records of the two databases being synchronized. The steps taken by this module are demonstrated in Fig. 3. The Parameters Table Generator 3 is responsible for creating a Parameter\_Table 4 which is used by all other modules for synchronizing the databases. Details of the Parameter\_Table are described in more detail below. The Synchronizer 15 has primary responsibility for carrying out the core synchronizing functions. It is a table-driven code which is capable of synchronizing various types of databases whose characteristics are provided in the Parameter\_Table 4. The Synchronizer creates and uses the Workspace 16, which is a temporary data array used during the synchronization process.

00752490-111396

A Translator 5 (A\_Translator) is assigned to the A\_database 13 and another Translator 9 (B\_Translator) to the B\_database 14. Each of the database Translators 5 and 9 comprises three modules: Reader modules 6 and 10 (A\_Reader and B\_Reader), which read the data from the databases 13 and 14; Unloader modules 8 and 12 (A\_Unloader and B\_Unloader), which analyze and unload records from the Workspace into the databases 13 and 14; and Sanitizing modules 7 and 11 (A\_Sanitizer and B\_Sanitizer), which analyze the records of the other database loaded into the Workspace and modify them according to rules of data value of its own database. In the preferred embodiment, the modules of the A\_Translator 5 are designed specifically for interacting with the A\_database 13 and the A\_Application 17. Their design is specifically based on the record and field structures and the rules of data value imposed on them by the A\_Application, the Application Program Interface (API) requirements and limitations of the A\_Application and other characteristics of A\_Database and A\_Application. The same is true of the modules of B\_Translator 9. These Translators are not able to interact with any other databases or Applications. They are only aware of the characteristics of the database and the Application for which they have been designed. Therefore, in the preferred embodiment, when the user chooses two Applications for synchronization, the Translation Engine chooses the two Translators which are able to interact with those Applications. In an alternate embodiment, the translator can be designed as a table-driven code, where a general Translator is able to interact with a variety of Applications and databases based on the parameters supplied by the Translation Engine 1.

Referring to Figs. 1, 2 and 3, the synchronization process is as follows. The Parameter\_Table 4 is generated

08752490-11396  
C

by the Parameter Table Generator 3. The Synchronizer 15 then creates the Workspace 16 data array and loads the History File 19 into the Workspace 16. The B\_Reader module 11 of the B\_Translator reads the B\_database records and  
5 sends them to the Synchronizer for writing into the Workspace. Following the loading of B\_Database records, the A\_Sanitizer module 8 of the A\_Translator 5 sanitizes the B\_Records in the Workspace. The A\_Reader module 7 of the A\_Translator 5 then reads the A\_Database records and sends  
10 them to the Synchronizer 16 for writing into the Workspace. The B\_Sanitizer module 12 of the B\_Translator 9 then sanitizes the A\_Records in the Workspace. The Synchronizer then performs the Conflict Analysis and Resolution (CAAR) on the records in Workspace. At the end of this analysis the  
15 user is asked whether he/she would like to proceed with updating the A\_ and B\_databases. If so, the B\_Unloader module of the B\_Translator unloads the appropriate records into the B\_database. The A\_Unloader module 6 then performs the same task for the A\_Database. Finally, the Synchronizer  
20 creates a new History File 19.

Fig. 3 is the pseudocode for the preferred embodiment of the Control Module 2 of the Translation Engine 1. Control Module 2 first instructs the Parameter Table Generator 3 of the Translation Engine 1 to create the  
25 Parameter\_Table (Step 100). *Figs. 4A and 4B are*  
*Fig. 4 is the* pseudocode for the preferred embodiment of the Parameter Table Generator module 3. The user is first asked to choose whether to use a previously chosen and stored set of preferences or to enter a new set of preferences (Step 150). Steps 151-165  
30 show the steps in which the user inputs his/her new preferences. In step 152, the user chooses whether to perform a synchronization from scratch or an incremental synchronization. In a synchronization from scratch,

08752490-111356

synchronization is performed as if this was the first time the two databases were being synchronized. In an incremental synchronization, the History File from the previous file is used to assist with synchronization. The user will likely choose incremental synchronization if there has been a prior synchronization, but the user may choose to synchronize from scratch where the user would like to start with a clean slate (perhaps due to significant change in the nature of the data in the databases). The user then selects the two Applications and related databases (A\_Database and B\_Database) to be synchronized (step 153). The user then chooses (step 154) whether the Synchronizer should use the default field mapping for those two databases during synchronization or the user will modify the field mapping. Field mapping is generally described in U.S. Patent No. 5,392,390 (incorporated by reference). In accordance with the user's preferences, the Parameter Table Generator then stores the appropriate A\_Database to B\_Database fields map (A→B\_Map) and B\_Database to A\_Database fields map (B→A\_Map) in the Parameter\_Table (Steps 155-158 and 159-163, accordingly).

If in step 150 the user selected to use previously chosen and stored set of preferences (steps 166-171), those preferences are loaded and stored in the Parameter\_Table (steps 169-170).

In case of date bearing records such as appointments and ToDo lists, the user enters the date range for which the user wants the records to be synchronized (step 172). The preferred embodiment allows the user to use relative date ranges (Automatic\_Date\_Range) (substeps 171 (a) and (b)). For example, the user can select the date range to be 30 days into the past from today's date and 60 days into the future from today's date. The Parameter Table Generator 3

then calculates and stores in the Parameter\_Table the Start\_Current\_Date\_Range and End\_Current\_Date\_Range values, the two variables indicating the starting point and the ending point of the date range for the current

5 synchronization session (step 173-174).

In steps 174 and 175, various parameters identifying the characteristics of the A\_Database and Application and B\_Database and Application are loaded from a database (not shown) holding such data for different Applications. These  
10 are in turn stored in the Parameter\_Table. One of the sets of parameters loaded and stored in the Parameter\_Table is the Field\_List for the two databases. The Field\_List\_A and Field\_List\_B contain the following information about each field in the data structure of the two databases:

- 15           1. Field name.  
            2. Field Type.  
            3. Field Limitations.  
            4. No\_Reconcile Flag.  
            6. Key\_Field Flag.  
20           7. Mapped\_Field Flag.

Field name is the name given to the field which the Translator for this Application uses. This name may also be the name used by the Application. Field Type identifies to the Synchronizer 15 the nature of the data in a field, e.g.,  
25 Data, Time, Boolean, Text, Number, or Binary. The Field Name does not supply this information to the Synchronizer. Field Limitations identifies the various limitations the database manager imposes on the contents of a field. These limitations include: maximum length of text fields, whether  
30 the text field must be in upper-case, range of permissible values (for example, in ToDo records priority field, the range of permissible values may be limited from 1 to 4), and whether a single line or multiple line field.

08752490-111396

No\_Reconcile flag indicates whether a field is a No\_Reconcile field, meaning that it will not be used to match records nor will it be synchronized although it will be mapped and possibly used in synchronization. Almost all  
5 fields will not be designated as No\_Reconcile. However, sometimes it is necessary to do so. Key\_Field flag indicates that a field should be considered as a key field by the Synchronizer 15.

10 Key fields are used by the Synchronizer in various stages of synchronization as will be discussed in detail below. The decision of identifying certain fields as key is based on examining the various Applications to be synchronized, their data structure, and the purpose for which the database is used. Such examination reveals which  
15 fields would best function as key fields for synchronization. For example, for an address book database, the lastname, firstname, and company name field may be chosen as key fields. For Appointments, the date field and the description field may be chosen as key fields.

20 Mapped\_Field flag indicates whether a field is mapped at all. The Synchronizer uses this flag to determine whether it should use the A→B\_Map or B→A\_Map to map this field. Unlike a No\_Reconcile field, an unmapped field will not be carried along through the synchronization.

25 Another set of parameters in the Parameter\_Table identify the Translator Modules 13, 14 for the two Applications which the user has selected. Because each Application is assigned its own Translator, it is necessary to identify to the Command Module and the Synchronizer which  
30 Translators should be used.

In step 102 of Fig. 1, the Translation Engine instructs the Synchronizer to load the History File. History File is the file which was saved at the end of last

08752490-1139E

synchronization. It contains the history of the previous synchronization which is necessary for use with the current synchronization in case of Incremental Synchronization. Records from the A\_Database and B\_Database are analyzed  
5 against the records of the history file to determine the changes, additions, and deletions in each of two databases since last synchronization and whether additions, deletions, or updates need to be done to the records of the databases. Referring to <sup>Figs. 5A and 5B,</sup> ~~Fig. 5,~~ in steps 200-201, the Synchronizer  
10 finds the appropriate History file to be loaded. If Synchronization\_from\_Scratch flag is set, the History File is deleted (step 203). If no History File is found, the synchronization will proceed as if it was a synchronization from scratch (step 204). If the Field Lists stored in the  
15 History File are not the same as the current Field Lists in the Parameter\_Table, or the mapping information is not the same, the synchronization will proceed as synchronization from scratch because the differences indicate that the History File records will not properly match the database  
20 records (steps 206-209).

In step 210, the Synchronizer uses the Field\_List for database B to create the Workspace 16. It is a large record array which the Synchronizer uses during synchronization. Referring to Fig. 2, Workspace 16 consist  
25 of two sections. First, the Synchronizer uses the Field List for the B\_Database to make a record array 21 which has all the characteristics of the B\_Database record structure. In addition, in each record in the Workspace, certain internal fields are added. One field is \_subtype containing  
30 Origin Tags. Two other fields, called Rep\_Basic and Rep\_Excl, are included for all Appointment and ToDo Sections. The Rep\_Basic field gives a full description of



the recurrence pattern of a recurring record. It includes the following parameters:

1. Basic\_Repeat\_Type
2. Frequency
- 5 3. StopDate
4. other parameters
5. Rep\_Excl

Basic\_Repeat\_Type contains the variable which indicates whether the recurring record is a daily, weekly, 10 monthly (same date each month), monthly by position (e.g., 3rd Friday of each month), yearly (e.g., July 4th each year), yearly by Position (e.g., 3rd Friday of September each year), quarterly, etc. This variable is set to No\_Repeat for non-recurring records.

15 Frequency indicates whether the pattern is, for example, for every week, every other week, etc. StartDate and StopDate show the first date and last date in the pattern. Some other parameters in the Rep\_Basic include, for example, a list of days to be included for the pattern 20 (e.g. I plan to hold a weekly staff meeting every Thursday starting November 15, 1997.)

Rep\_Excl is the exclusion list. It is a list of dates which at some point belonged to the recurring record, but have since been deleted or modified and no longer are an 25 event represented by the recurring record.

Since some databases do not provide for recurring types of records, the synchronization process sometimes must create single records for each of the instances of a recurring record for those databases. For example, for a 30 recurring lunch every Thursday, the synchronization must produce a single record for each Thursday in such a database. This is accomplished by the process of fanning which uses Rep\_Basic. Each of those instances is called a

fanned instance. Fig. 6 sets out the preferred embodiment of the process of fanning a record.

Fanning of recurring records also takes into account another set of considerations regarding date range

5 limitations and usefulness of instances to the user.

First, fanning is limited to the applicable date range. Second, the number of fanned instances is limited.

When synchronizing Databases A and B, the preferred embodiment permits different sets of limits on fanned  
10 instances to be established for each Database. This, for example, assists with managing storage capacity of a memory-constrained handheld device when being synchronized with a database on a desktop PC.

If the current Date Range is large enough to  
15 accommodate more than the maximum number of instances which might be generated, those instances will be chosen which are likely to be most useful to the user. In the preferred embodiment, it is assumed that future instances are always more useful than past instances, that near future instances  
20 are more useful than distant future instances, and that recent past instances are more useful than distant past instances. Therefore, based on these assumptions, a fanning date range is calculated (Fig. 6, step 236).

Referring to Fig. 2, in the second step of creating  
25 the Workspace, the Synchronizer establishes an Extended Index Array 20 which has an index entry associated with each entry in the record array. Each index contains the following variables:

1. Next\_In\_CIG:
- 30 2. Next\_In\_SKG:
3. Next\_In\_FIG
4. Key\_Field\_Hash
5. A\_Unique\_ID\_Hash

6. B\_Unique\_ID\_Hash
7. Non\_Key\_Field\_Hash
8. Non\_Date\_Hash
9. Exclusion\_List\_Hash
- 5 10. Start\_Date&Time
11. End\_Date&Time
12. Various bit flags

Next\_In\_CIG is a linkage word, pointing to next member of the same Corresponding Item Group (CIG). A CIG is  
 10 a group of records, one from each database and the History File, if applicable, which represent the same entry in each of the databases and the History File. There may be one, two or three records in a CIG. Next\_In\_SKG is a linkage word, pointing to next member of the Same Key Fields Group  
 15 (SKG). An SKG is a group of records having the same key fields. Next\_In\_FIG is a linkage word, pointing to the next member of the Fanned Instances Group (FIG). A FIG is the group of fanned instances which correspond to a single recurring record.

20 Key\_Field\_Hash is hash of all Key\_Fields.  
 A\_unique\_ID\_Hash is hash of unique ID, if any, assigned by A\_Database. B\_unique\_ID\_Hash is hash of unique ID, if any, assigned by B\_Database. Non\_Key\_Field\_Hash is hash of all Non-Key Match Field, a Match Field being any mapped field  
 25 which is not flagged as No\_Reconcile. Non\_Date\_Hash is hash of all Non-Date Non-Key Match Fields. Exclusion\_List\_Hash is hash of recurring record's exclusion list.

Start\_Date&Time and End\_Date&Time are used for Appointment and ToDo type record only, indicating the start  
 30 and end date and time of the record. They are used to speed up comparing functions throughout the synchronization. Hash values are also used to speed up the process of comparison. The preferred embodiment uses integer hashes.

Hash value computation takes into account certain rules of data value for fields, as will be described in more detail below.

5 In the preferred embodiment, the record array 21 is stored on magnetic disk of a computer whereas the Extended Index 20 is held resident in memory. The Extended Indexes have record pointer fields which point to each of the records on the disk file.

10 The Control Module 2 now instructs the synchronizer to load the History File into the Workspace (Fig. 3, step 102). Referring to Fig. 6, the synchronizer loads the records beginning in first available spot in the Workspace (step 211). The Synchronizer then performs an analysis on each of the records and resets some of the values in the  
15 records (steps 212-228). The records are also checked against the current date range and those falling outside of it are marked appropriately for Fast synchronization function, which will be described below. In case of recurring records, if any of the instances is within the  
20 current date range, then the recurring record itself will be considered within the current date range (steps 217-227).

25 The synchronizer then builds SKGs by finding for each history record one record which has matching key fields and by placing that record in the SKG of the history record (step 215-216). Referring to Fig. 7, steps 250-258 describe the Key\_Field\_Match function used for matching records for SKG.

30 When comparing two records or two fields, in the preferred embodiment, the COMPARE function is used. The COMPARE function is intelligent comparison logic, which takes into account some of the differences between the rules of data value imposed by the A\_Application and the B\_Application on their respective databases. Some examples

are as follows. The COMPARE function is insensitive to upper and lower case letters if case insensitive field attribute is present. Because some Applications require entries to be in all capital letter, the COMPARE function  
5 ignores the differences between upper and lowercase letters. The COMPARE function takes into account any text length limitations. For example, when comparing "App" in the A\_Database and "Apple" in the B\_Database, the COMPARE function takes into account that this field is limited to  
10 only 3 characters in the A\_Database. It also takes into account limits on numerical value. For example, priority fields in the A\_Application may be limited to only values up to 3, whereas in the B\_Application there may not be any limitation. The COMPARE function would treat all values in  
15 B\_records above 3 as 3.

The COMPARE function may ignore various codes such as end of line characters. It may strip punctuation from some fields such as telephone numbers and trailing white space from text fields (i.e "Hello " is treated as  
20 "Hello"). It also considers field mapping. For example, if the only line that is mapped by the A-B\_Map is the first line of a field, then only that line is compared. When comparing appointment fields, because different databases handle alarm date and time differently when Alarmflag is  
25 false, the COMPARE function treats them as equal even though the values in them are not the same. It skips Alarm Date and Time, if the Alarm Flag is False. It also ignores exclusion lists when comparing recurring records.

In an alternate embodiment, the COMPARE function may  
30 take into account more complicated rules for data value of the two Applications, such as the rules for data value imposed by Microsoft Schedule+, described above. Such a COMPARE function may be implemented as a table driven code,

the table containing the rules imposed by the A\_Application and the B\_Application. Because the COMPARE function has a specific comparison logic and takes into account a number of rules, the hashing logic must also follow the same rules.

- 5 It should be noted that the COMPARE function is used throughout the preferred embodiment for field comparisons.

✓ Now that the History File is loaded into the Workspace, the Control Module 2 instructs the B\_Translator 13 to load the B\_Database records (Fig. 3, step 103).

- 10 Referring to Fig. 8, steps 300-308, the B\_Reader module 11 of the B\_Translator 13 loads each B\_record which has the right Origin Tag, which will be explained in more detail below.

- 15 The record must also be within the loading date range, which is a concatenation of the previous and current date ranges. The B\_Translator sends these records to the Synchronizer which in turn stores them in the Workspace. When synchronizing with a date range limitation, all records which fall within either the previous or the current date ranges are loaded. The current date range is used during unloading to limit the unloading of the records to only those records which fall within the database's current date range. In an alternate embodiment of the invention, each database or Application can have its own date range for each
- 20
- 25 synchronization.

- Most Applications or databases permit record-specific and field-specific updates to a Database. But some Applications or databases do not. Instead the Translator for these Application must re-create the whole database from scratch when unloading at the end of synchronization. These
- 30
- databases are identified as Rebuild\_All databases. To accommodate this requirement all records from such a database must be loaded into the Workspace, so that they can

later be used to rebuild the whole database. These databases records, which would otherwise have been filtered out by the date range or the wrong origin tag filters, are instead marked with special flag bits as Out\_Of\_Range or  
5 Wrong\_Section\_Subtype. These records will be ignored during the synchronization process but will be written back unmodified into the database from which they came by the responsible Unloader module 6, 10.

Control Module 2 next instructs the A\_Translator 5  
10 to sanitize the B-records. Referring to Fig. 9, steps 350-361, the A\_Sanitizer module 8 of the A\_Translator 5 is designed to take a record having the form of an A\_Record and make it conform to the specific rules of data value imposed by the A\_Application on records of the A\_Database.  
15 A\_Sanitizer is not aware which database's field and records it is making to conform to its own Application's format. It is only aware of the A\_Application's field and record structure or data structure. Therefore, when it requests a field from the sanitizer using the A\_Database field name, it  
20 is asking for fields having the A\_Database data structure. The Synchronizer, in steps 375-387, therefore maps each record according to the B→A\_Map. In turn, when the Synchronizer receives the fields from the A\_SANITIZER, it waits until it assembles a whole record (by keeping the  
25 values in a cache) and then maps the record back into the B format using the A→B\_Map.

How a record or a field is sanitized in step 354 and 357 depends on the rules of data value imposed by the A\_Application. For example, all of the logic of intelligent  
30 comparison in the COMPARE function described above can be implemented by sanitization. However, sanitization is best suited for more complex or unique types of database rules for data value. For example, consider the Schedule+ rules

regarding alarm bearing Tasks records described above. Fig. 10 shows a sanitization method for making records of incompatible databases conform to the requirements of Schedule+. Without sanitization, when a Tasks record of a  
5 Schedule+ database is compared to its corresponding record in another database, the Tasks record may be updated in fields which should be blank according to the Schedule+ rules of data value. Such an update may possibly affect the proper operation of Schedule+ after synchronization.

10 Referring to Fig. 11, following sanitization of all B\_Records into the Workspace, the Synchronizer sets the values for the Extended Index of each record based on the record's values (steps 451-459). Also if the records in the B\_Database bear a unique ID, and matches for those unique  
15 IDs are found in the H\_Records in the Workspace, the two records are joined in a CIG because they represent the same record in both History File and B\_Database (step 462). The record is also joined to an SKG it may belong to (step 464). The loading of B\_Records is now complete.

20 The Control Module 2 of the Translation Engine 3 now instructs the A\_Translator 5 to load the records from the A\_Database (step 105). The loading process for the A\_Records is the same as the loading process for the B\_Database, except for some differences arising from the  
25 fact that records in the Workspace are stored according to the B\_Database data structure. Therefore, as the synchronizer 15 receives each A\_record from the A\_Reader module 7 of the A\_Translator 5, the Synchronizer maps that record using the A→B\_Map before writing the record into the  
30 next available spot in the Workspace. Since the A\_records are mapped into the B\_Record format, when the B\_Sanitizer is instructed by the Control Module 2 to begin sanitizing those records and starts asking for them from the synchronizer,



they already have the B\_Database format. Therefore, the synchronizer 15 does not need to map them before sending them to the B\_Sanitizer module 12 of the B\_Translator 19. For the same reason, there is no need for them to be mapped once they are sent back by the B\_Sanitizer after having been sanitized. Once all the records are loaded, the records will undergo the same orientation analysis that the B\_Records underwent (Fig. 11).

At this point, all records are loaded into the Workspace. SKGs are complete since every record at the time of loading is connected to the appropriate SKG. CIGs now contain all records that could be matched based on unique IDs. At this point, the records in the Workspace will be analyzed according to Conflict Analysis and Resolution ("CAAR") which is set out in Fig. 12 and in more detail in Figs. 13-18 and corresponding detailed description.

First, in step 500, ID bearing fanned instances in the History File records are matched to the fanned instances in the ID bearing database from which they came. The records from the database which have remained unchanged are formed into a new FIG. A new Synthetic Master is created based on those records and joined to them. The records which have been changed or deleted since last synchronization are set free as single records. They also result in a new exclusion list being created based on an old exclusion list and these new single records.

Second, in step 501, matches are sought for the ID based CIGs which are the only CIGs so far created in order to increase the membership of those CIGs. Preferably an exact all fields match is sought between current members of a CIG and a new one. Failing that, a weaker match is sought.

Third, in step 502, master/instances match is sought between recurring records and non-unique ID bearing instances by trying to find the largest group of instances which match certain values in the Recurring Master.

5 Fourth, in step 503, the items remaining in the SKGs are matched up based on either exact all field match or master/instance match, or a weaker match.

10 Fifth, in step 501, the appropriate CIG\_Types are set for all the CIGs. CIG\_Types will determine what the outcome of unloading the records will be.

15 Referring to Fig. 13, first step in CAAR is analyzing unique ID bearing Fanned Instance Groups. This analysis attempts to optimize using unique IDs assigned by databases in analyzing fanned instances of recurring records.

20 The analysis is performed for all Recurring Masters (i.e. all recurring records) which have ID-bearing fanned instances (or FIG records) in the H\_File (step 550). All FIG records in the History File associated with a Recurring Master are analyzed (steps 551-559). They are all removed from the SKG. If a FIG record is a singleton CIG, it means that it was deleted from the database since the previous synchronization. Therefore, it is added to the New\_Exclusion\_List (step 553). If a FIG record is a  
25 doubleton and is an exact match, it means that the record was not modified since the previous synchronization. In this case, the record from the database is also removed from SKG (step 555). If a FIG record is a doubleton but is not an exact match for its counterpart in the database, it means  
30 that the record was changed in the database. The History File record is treated as a deletion and therefore added to the New\_Exclusion\_List. The modified record in the database, which does not match the recurring record any

longer, is treated as a free standing record un-associated with the Recurring Master (step 557).

5 Upon analysis of all FIG records, a new record, the Synthetic Master, is created and joined in a CIG with the Recurring Master (step 231-236). The Synthetic Master has the same characteristics as the Recurring Master, except that it has a new exclusion list which is a merger of the New\_Exclusion\_List and the Exclusion\_List of the Recurring Master (step 563). Also a new FIG is created between the  
10 Synthetic Master and the CIG-mates of all FIG records from the History File (step 565).

15 In steps 567-569, the Synchronizer checks to see if there are some instances of the Recurring Master which fall within the previous synchronization's date range but fall outside of the current synchronization's date range. If so, the Fan\_Out\_Creep flag is set, indicating that the date range has moved in such a way as to require the record to be fanned for the database before unloading the record. The Fan\_Out\_Creep flag is an increase in the value in the  
20 Non\_Key\_Field Hash of the Recurring Master. In this way, the Recurring Master during the unloading of the records will appear as having been updated since the last synchronization and therefore will be fanned for the current date range.

25 In step 570, all the FIG records analyzed or created in this analysis are marked as Dependent\_FIGs. This results in these records being ignored in future analysis except when the recurring records to which they are attached are being analyzed.

30 At the end of the above analysis, all the records having a unique ID assigned by their databases have been matched based on their unique ID. From this point onward, the records which do not have unique IDs must be matched to

03752490-111396  
SECRET

other records based on their field values. In the preferred embodiment, there are two categories of field value matches: strong matches and weak matches. A strong match between two records that have matching key fields is when non-key fields of the two records match or it is a Recurring Master and a fanned instance match (Fig. 14, steps 606-610). Referring to Fig. 15, a weak match between two records that have matching key fields is when the following are true: each of the two records are from different origins, because two records from the same source should not be in a CIG (e.g., A\_Database and History File); each is not a weak match for another record because there is no reason to prefer one weak match over another; each is not a Dependent\_FIG since these records do not have an independant existence from their recurring masters; both records are either recurring or non-recurring since a recurring and a nonrecurring should not be matched except if one is an instance of the other in which case it is a strong match; and, in case of non-recurring, they have matching Key\_Date\_Field which is the same as the Start\_Date in the preferred embodiment because items on the same date are more likely to be modified versions of one another.

Referring to Fig. 14, these two types of matching are used to match records to existing CIGs for History File records which have been created based on matching unique IDs. Only doubleton CIGs are looked at, because singleton CIGs are handled in step 504 of Fig. 12 and tripleton CIGs are complete (steps 601-604). If a strong match is found, then if the record was a weak match in another CIG, it is removed from that CIG, and new weak match is found for that CIG (612-614). While weak matches are left in SKGs in case they will find a strong match, strong matches are removed from their SKGs (step 614). If a strong match is not found,

C

[illegible]

then marked with Fan\_Out\_Creep flag, if necessary (step 685).

The next step in CAAR is completing analysis of records in SKGs (Fig. 12, step 504). Referring to Fig. 17, this analysis attempts to increase the population of CIGs up to a maximum by finding key field based matches with records from a source different from those of the CIG records. This analysis is performed by analyzing all the records in the SKGs except for the singleton SKGs (steps 703 and 712). The first thing is to remove any members that have already been marked as WEAK matches attached to ID-based doubleton CIGs. Those are left in the SKG up to this point to allow for the possibility that a STRONG match would be found instead. But that is not possible any longer (steps 713-715). Once the weak matches have been removed, all remaining SKG members belong to singleton CIGs. Any non-singleton CIGs which are formed from here on will be purely key field based.

Throughout the remaining SKG Analysis we are careful not to seek H\_Record-A\_Record or H\_Record-B\_Record matches for unique ID-bearing Source, since that would violate the exclusively ID-based matching scheme that applies in such cases. Note however that an A\_Record-B\_Record match is acceptable even if both A\_Database and B\_Database are unique ID-bearing databases.

Given that Key Field should not be performed where ID based matches are available (or otherwise there may be matches between records with differing IDs), there are limits to how big CIGs can get at this point. If both A and B\_Databases are unique ID-bearing, any remaining H\_Record must remain in Singleton CIGs, because they are prohibited from forming key fields based matches with items from either databases. Such H\_Records are simply removed from the SKG when they are encountered. If just one of the two databases

00752490-11396  
SECRET 0642780

being synchronized is unique ID-bearing then the maximum population that any CIG can now attain is 2 (Fig. 18, steps 750-751). If neither database is unique ID bearing then the CIG\_Max\_Size is three. For every CIG which is analyzed in  
5 Fig. 17, the CIG\_Max\_Size is set according to this logic. When a CIG reaches its maximum possible population all of its members are removed from the appropriate SKG.

First, strong matches for the H-records are searched for, before trying to find A-B matches. If both Databases  
10 are non-unique ID-bearing then two strong matches for each H\_Record, an H-A and an H-B match, are sought (steps 715-720). If finding a strong match results in reaching the CIG\_Max\_Size, all members of the CIG are removed from the SKG (step 721).

15 When maximum CIG population is 3, weak matches are sought for strong matching CIG doubleton in order to build triplet CIGs. The first weakly matching SKG member is added to the CIG (steps 722-728). Whether or not a weak match is found for any of the doubleton CIGs, its members are removed  
20 from the SKG (step 726). As there are no strong matches left in the SKG, weak matches are found for any remaining SKG members and joined to them in CIGs (steps 722-725).

At this stage, all CIGs are built. They must now be examined to determine what needs to be done to these records  
25 so that the databases are synchronized, i.e. whether the records in the CIGs need to be added, deleted or changed in the two databases. First step is determining the CIG\_TYPE which represents the relation between the records. The following CIG types are defined, all using a 3-digit number  
30 that represents values found for A\_DATABASE, History File, and B\_Database, respectively:

1. 001 - record is "new" in the B\_DATABASE

06752490-111396

2. 010 - record is present in History, but absent in both A\_Database and B\_Databases
3. 100 - record is "new" in the A\_Database
4. 101 - record is "new" in both A\_Database and B\_DATABASE; same in both
5. 102 - record is "new" in both A\_Database and B\_DATABASE; different in each (conflict)
6. 110 - record deleted from B\_DATABASE
7. 011 - record deleted from A\_Database
- 10 8. 012 - record deleted from A\_Database and changed on B\_DATABASE (DEL vs CHANGE conflict)
9. 210 - record changed on A\_Database and deleted from B\_DATABASE (DEL vs CHANGE conflict)
10. 111 - record unchanged since previous synchronization
- 15 11. 112 - record changed on B\_DATABASE only since previous synchronization
12. 211 - record changed on A\_Database only since previous synchronization
- 20 13. 212 - record changed identically on both since previous synchronization
14. 213 - record changed differently on each since previous synchronization (conflict)
15. 132 - a conflict (102 or 213) was resolved by forming a compromise value; Update both
- 25 16. 13F - created when a 132 Update both CIG is Fanned into the B DATABASE
- Figs. 19A and 19B show*  
*Fig. 19 shows the method used for setting all*
- except the last two CIG\_Types which are set in other operations.
- 30

Four of the CIG types assigned above involve conflicts: 102, 213, 012, and 210. Conflicts are those instances where a specific conflict resolution rule chosen

21



03752490-11396  
9677 062580

by the user or set by default, or the user's case by case decision, must be used to determine how the records from the databases should be synchronized. CIG types 012 and 210 are cases where a previously synchronized record is changed on one side and deleted on the other. In the preferred embodiment, such conflicts are resolved according to the rule that CHANGE overrules the DELETE. So the net result for CIG type 012 is to add a new record to the A\_Database to match the record in the B\_DATABASE. The reverse is true for CIG type 210, where a new record is added to the B\_Database. In an alternate embodiment, the user may be allowed to register an automatic preference for how to resolve such conflicts or decide on a case-by-case basis a conflict resolution option.

The other two conflict types -- 102 and 213 -- are resolved in the preferred embodiment according to the Conflict Resolution Option established by the user. First, the user may choose to ignore the conflict. This option leaves all 102 and 213 conflicts unresolved. Every time synchronization is repeated the conflict will be detected again and ignored again, as long as this option remains in effect and as long as the conflicting records are not changed by other means.

The user may choose to add a new record to each of the two databases. This option resolves 102 and 213 conflicts by adding the new A\_Record to the B\_Database, and adding the new B\_Record to the A\_Database. This option is implemented by breaking a 102 CIG into two separate CIGs (types 100 and 001) and a 213 CIG into three separate CIGs (types 100, 010, and 001). Subsequent processing of those descendant CIGs causes new records to be added across and stored in the History File.

32

03752490-11396  
SECRET 0542580

The user may elect that A\_Database records should always trump or win over B\_database records. This option is implemented by changing the CIG type to 211 - the processing during unloading the records changes the record value in the B\_Database to match the current record value in the A\_Database.

The user may elect that B\_Database records should always trump or win over B\_database records. This option is implemented by changing the CIG type to 112 - the processing during unloading the records changes the record value in the A\_Database to match the current record value in the B\_Database.

The user may choose to be notified in case of any conflict. The user is notified via a dialog box 30, shown in Fig. 20, whenever a CIG type conflict of 102 or 213 arises. The dialog box shows the record that is involved in the conflict 31. It also shows the A\_Database 32 and B\_Database 33 values for all conflicting fields, in a tabular display, with Field Names appearing in the left column 34. A dropdown list (not shown) in the lower left hand corner of the dialog 37, offers a total of three choices - add, ignore, and update. The use may choose to add new records or ignore the conflict. The user may also choose that the A\_Record or B\_Record should be used to update the other record. The user may also decide to create a compromise record by choosing values of different fields and then choosing update option. In this case, the CIG type is changed to 132, which results in an updating both databases with the new record compromise record.

When the user has chosen to be notified in case of conflict, if the user chooses to ignore conflict or that either the record of the A\_Database or the B\_DATABASE should win, the CIG type is left as a conflict CIG type (102 or

33

213) and a separate Conflict Resolution Choice is stored in the FLAGS word associated with each CIG member.

The final step in setting CIG\_Types is the process for dealing with difficulties which arise from exclusion  
5 lists. For example, in a triple Recurring Master CIG, suppose the History File Recurring Master does not have any excluded instances. The A\_Record has the following exclusion list:

12/1/96, 12/8/96

10 The B\_Record has the following exclusion list:

1/1/97, 1/8/97, 1/15/97, 1/22/97, 1/29/97

If comparison of the Recurring Masters includes comparing exclusion list Field Values, this set of changes would cause the Synchronizer to report a CIG type 213  
15 conflict.

If the Conflict Resolution Option is set to A\_Database record wins, then the outcome prescribed by the Synchronizer would be for the A\_Database to keep its exclusion list as is and for the B\_Database to make its  
20 exclusion list match that of the A\_Database.

The result would be to have a lot of duplicate entries in both Databases. The A\_Database would have five duplicate entries in January 97 - that is the five unmodified Recurring Master instances, plus the five  
25 modified instances added across from B\_Database to A\_Database. The B\_Database would have five duplicate entries in January 97, since synchronization has wiped out the five exclusions that were previously recorded in the B\_Database exclusion list.

30 Two steps are implemented for dealing with this problem. First, the COMPARE function does not take into account exclusion list differences when comparing recurring records. Second, referring to Fig. 21, any new exclusions

03752450-111396  
SECRET

added on to one recurring record will be added to the other record. The merging of exclusion lists is done regardless of any updates or conflicts, even unresolved conflicts, between the A\_Database and B\_Database copies of a Recurring Master. One exception is for CIG type 102 conflict which is left unresolved where Exclusion lists are not merged, because the user has chosen to leave those records as they are.

10 In most cases where it is necessary to merge exclusion lists, the CIG types and/or the Conflict Resolution Choice to arrange for all necessary updates to be performed during the unloading phases of synchronization.

15 First, A\_Database and B\_Database records' exclusion lists are compared. In case of databases which do not permit recurring items, the exclusion list of the Synthetic Master is compared to the recurring record of the other database (step 852). If there is no difference, then nothing is done (step 853). If there are differences, then it is determined which exclusions appear only in one record. 20 This comparison always yields one of the following scenarios: (1) all one-side-only Exclusions are on the A\_Database (so Exclusions should be added to the B\_Database); (2) all one-side-only Exclusions are on the B\_Database (so Exclusions should be added to the A\_Database); and (3) there are one-side-only Exclusions on both sides (so Exclusions should be added to both databases).

30 In each of these cases a separate table is used to look up instructions, for how to handle each specific situation (Figs 22-24). The tables cover all possible combinations of previous CIG types and outcome codes with all possible exclusion list changes (new and different exclusions added on A\_Database, or on B\_Database, or on both

sides). Fig. 22 table is used in case of scenario 1. Fig. 23 table is used in case of scenario 2. Fig. 24 table is used in case of scenario 3 (Fig. 21 steps 854-856).

The analysis of records is now complete, and the records can be unloaded into their respective databases, including any additions, updates, or deletions. However, prior to doing so, the user is asked to confirm proceeding with unloading (Fig. 3, step 108-109). Up to this point, neither of the databases nor the History File have been modified. The user may obtain through the Translation Engine's User Interface various information regarding what will transpire upon unloading.

If the user chooses to proceed with synchronization and to unload, the records are then unloaded in order into the B\_Database, the A\_Database and the History File. The Unloader modules 6,10 of the Translators 5,9 perform the unloading for the databases. The Synchronizer creates the History File and unloads the records into it. The Control Module 2 of the Translation Engine 1 first instructs the B\_Translator to unload the records from Workspace into the B\_Database. Referring to <sup>Figs. 25A and 25B</sup> Fig. 25 for each CIG to be unloaded (determined in steps 902-907), based on the CIG\_TYPE and which database it is unloading into (i.e., A or B), the unloader looks up in the table in <sup>Figs. 26A-26D</sup> Fig. 26 the outcome that must be achieved by unloading - that is, whether to update, delete, add, or skip (Leave\_Along) (step 908). In steps 909-913, the unloader enforces date range restriction for a database subject to date range. The user may select, or a selection may be made by default, whether to enforce the date range sternly or leniently. In case of stern enforcement, all records outside of the current date range would be deleted. This is useful for computers with

small storage capacity. In case of lenient enforcement, the records are left untouched.

Based on the result obtained from looking up the unloading outcome in the table, the unloader then either  
5 adds a new record (steps 920-926), deletes an existing record (steps 914-919), or updates an existing record (steps 927-933). It should be noted that because we only update those fields which need to be updated (step 928), the fields which were sanitized but need not be updated are not  
10 unloaded. Therefore, the values in those fields remain in unsanitized form in the database.

Referring to step 914, in some Applications when a Recurring Master must be added or updated, the record may have to be fanned out despite the ability of the Application  
15 to support recurring records. For example, the Schedule+ Translator is generally able to put almost any Recurring Master Item into Schedule+ without fanning, but there are some exceptions. The Schedule+ Translator uses one Schedule section to handle all appointments and events. For  
20 appointments, almost any recurrence pattern is allowed, but for events the only allowable true repeat type is YEARLY. DAILY recurring events can be dealt with by being translated into Schedule+ multi-day events which are not recurring but extend over several days by setting the EndDate some time  
25 after the Start Date. But for the DAILY case there are restrictions. In particular exclusions in the midst of a multi-day Schedule+ event cannot be created. So the Translator decides that if section type is ToDos or the item is a non-Event Appointment, then the record need not be  
30 fanned out. But if item is a YEARLY or DAILY with no exclusions then it can be stored as a Schedule+ yearly or daily event. Otherwise, it must be fanned.

37

*Figs. 27A and 27B,*

C Referring to ~~Fig. 27~~, steps 950-984 set out the preferred embodiment of fanning recurring records that must be updated. All cases fall within three scenarios, shown in Fig. 29.

5 In the first scenario a record which is a Recurring Master, and its counterpart in the other database is a Recurring Master, must be fanned now for its own database (steps 951-959). If the CIG\_TYPE of the record is 132 (i.e. update both records), then it is changed to 13F which is a  
10 special value specifically for this situation (step 951). For other CIG\_Types, the CIG is broken into three singleton and given CIG\_Types signifying their singleton status. In both of these cases, the function Fanning\_For\_Add (steps 986-996, described below) is called.

15 In the second scenario, the record was fanned previously and is going to be fanned now also. First, the dates of the instances are recorded in a temporary date array (steps 961-963). This array is compared to an array of the fanned instances of the recurrence pattern of the CIG  
20 Recurring Master from the other database (steps 965-966). The dates which are not in the array of fanned instance are marked for deletion (step 967). The dates which are not in the temporary date array should be added to the unloading databases and therefore new FIG records are created for  
25 those dates (steps 968-973). The dates which appear in both arrays are compared to the Synthetic Master and marked accordingly for UPDATE or Leave\_Alone (steps 974-978).

In the third scenario, the record which was previously fanned should now be fanned also. The opposing  
30 database's record in this scenario is also fanned instances. This is perhaps the most peculiar of the three cases. For example, a database may be able to handle multi-day (i.e. daily recurring) records but not any exclusion dates for

such items. Such database may be synchronized with another database which fans all records in the following manner. A record representing a 7-day vacation in the Planner section of the database is fanned out to form 7 individual vacation  
5 days in the other database. One instance is deleted in the other database. Upon synchronizing the two databases, b/c  
✓ the first databases does not does not provide for exclusion lists, the record must now be fanned.

10 In this scenario, Master Records in a CIG are marked as Garbage. Any FIG members attached to the H\_Record, if any, are also marked as Garbage. All Instances found in the opposing database's FIG are truned to singleton CIGs with CIG type 100 or 001 so that they will be added to the  
15 unloader's database when unloading is done. In this way the instances from one database is copied to the database providing for recurring records.

Steps 985-995 describe the Fanning\_For\_Add Function which is used when outcome is to update or when the function is called by the Translator fanning for update. For each  
20 instance generated by fanning out the recurring record, a clone of the Recurring Master is created but excluding Rep\_Basic and Rep\_Excl field values and the unique ID field. All adjustable Date Fields (e.g. Start Date, End Date, and Alarm Date) are set and hash values for the new record is  
25 computed. The new record is then marked as Fanned\_For\_A or Fanned\_For\_B, as the case may be. This is then attached to the Recurring Master Item as a FIG member.

Following unloading of the B\_RECORDS, the Control Module 2 instructs the A\_Translator to unload the A\_Records  
30 from the Workspace (Fig. 3, step 111). This unloading is done in the same way as it was done by the B\_Translator. In case of Rebuild\_All Translators which have to reconstruct the database, all records which were loaded from the



database but were not used in synchronization are appended and unloaded as the Translator builds a new database for its Application.

5 The Control Module 3 next instructs the Synchronizer to create a new History File (step 112). Referring to Fig. 28, for every CIG in the Workspace, it is first determined which record should be unloaded to History File (steps 1001-1003). In the next step, Excl\_Only flag is checked, which is set by the Merge\_Exclusion\_List logic (Fig. 21-24). If  
10 that flag is set, a new record for unloading is created which has all fields taken from the History File record, except that the newly merged exclusion list is inserted into that record (step 1004). Before storing the record in the History File, all Flag Bits in the Extended Index are  
15 cleared except the bit that indicating whether or not this is a recurring item (step 1005). The item is marked as a History File record to indicate its source. The CIG, FIG, and SKG are reset. All the HASH values and  
20 Start&EndDate&Time will be stored. All applicable unique ID are also stored (Steps 1006-1009). The current record is then stored in the new History File (step 1010). If the current record is a Recurring Master for an ID-bearing FIG, we now store the whole FIG (i.e. all Fanned Instances) in the History File, with the FIG linkage words set in the  
25 History File to hold the FIG records together (step 1011). Fanned instances which do not bear unique IDs are not stored in the History File since they can be re-generated by merely fanning out the Recurring Master.

30 Once all records are unloaded, various information necessary for identifying this History File and for the next synchronization are written into the History File (step 1013).

At this point Synchronization is complete.

03752490-111396  
96ET 06425290

Applications, such as scheduling Applications, often have more than one database. Each of these databases are known as sections. Each of these sections contain different data and must be synchronized with their corresponding sections in other Applications. However, there is not necessarily a one to one relationship between sections of various Applications. For example, Application A may comprise of the following sections: Appointments, Holidays, Business Addresses, Personal Addresses, and ToDo.

10 Application B however may comprise of the following sections: Appointments, Addresses, ToDo-Tasks, and ToDo-Calls. Although the general character of the sections are the same, there is not a one to one relation between the sections of these two Applications: Appointments and

15 Holidays in A contain the same type of data as Appointments in B; Business Addresses and Personal Addresses in A contain the same type of data as Addresses in B; and ToDo in A contains the same type of data as ToDo-Tasks and ToDo-Calls in B. Therefore, when synchronizing the sections of these

20 two Applications, it is necessary to synchronize at least two sections of one Application with one section of another Application.

The preferred embodiment performs this type of synchronization by providing for a number of section

25 categories: Appointment, ToDo, Note, Address, and General Database. All sections of a particular Application are studied and categorized according to this categorization. Therefore, in the above example of Application A, Appointments and Holidays are categorized Appointment type

30 sections (or database), Business Address and Personal Address as Address type sections, and ToDo as a ToDo type section.

For creating the map for mapping sections onto each other, an exact section match is always sought between sections of the two Applications. If not, one of the sections which were categorized as a section type is chosen to be the Main\_Section among them. Other sections of the same type are referred to as subsections. All databases of the same type from the other Application will be mapped onto the Main\_Section.

To properly synchronize from one time to the next, it is necessary to keep track of the source of records in the Main\_Section. In the preferred embodiment, if a record in the Main\_Section of the A\_Application does not come from the Main\_Section of the B\_Application, one of fields in the record, preferably a text field, is tagged with a unique code identifying the subsection which is the source of the record. This is the record's Origin Tag. All records in the Workspace and the History File include a hidden internal field called \_subType which contains the unique subsection code. Main\_Section's field value in the preferred embodiment is zero so that it will not be tagged. When a record is loaded from a database into the Synchronization Workspace, the tag is stripped from the TagBearer field and put in the \_subType field. If there is no tag, then the \_subType is set to be the subType of the present section. If the TagBearer field is mapped then when reading records into the Workspace the tag, if any, is stripped from the TagBearer field value place it in \_subtype.

Conversely when unloading records from the Workspace to a Database, the TagBearer field is tagged by a tag being added if the record is not from the Main\_Section.

A Fast Synchronization database is a database which provides a method of keeping track of changes, deletions, and additions to its records from one synchronization to the

03752450-11396  
SECRET

next. These databases speed up the synchronization process because only those records which have been modified need to be loaded from the database. Since the majority of records loaded by regular Translators are unchanged records, far  
5 fewer records are loaded from the database into the Synchronizer.

Certain features are required for a database to be a Fast Synchronization database. The database records must have unique IDs and must have a mechanism for keeping track  
10 of which records are added, changed, or deleted from synchronization to synchronization, including a list of deleted records. Unique IDs are required to accurately identify records over a period of time.

There are at least two ways to keep track of  
15 additions, changes, and deletions in a database.

First, some databases maintain one Dirty bit per record which is a boolean flag that is set when a record is created or modified and is cleared when a function for clearing Dirty bits is called. Some databases offer a Clear  
20 DirtyBit function that clears the bit of an individual record. Other databases offer a ClearDirtyBits function that clears the Dirty bits of all records in a database. The record-specific ClearDirtyBit function allows the preferred embodiment to use the database itself to keep  
25 track of additions and changes.

The global ClearDirtyBits function forces the preferred embodiment to clear all Dirty bits at the conclusion of every Synchronization. Then as database edits are made by the user in between synchronizations, the  
30 affected records are marked as Dirty. When Synchronization is performed again, only the Dirty records are loaded.

Second, some databases maintain a Date&Time stamp of when the record was added or last time the record was

modified. A Translator for such a database finds all records which were added or modified since the previous synchronization by searching for Date&Time stamps more recent than the Date&Time of the Last Synchronization.

5 A Fast Synchronization database must also keep track of deletions. This is done by maintaining a list of deleted records which can be read by a Translator.

A Translator sending Fast Synchronization database records to the Synchronizer provides only records which have  
10 been changed, deleted, and added since the previous synchronization. Therefore, unlike a regular database Translator, a Fast Synchronization Translator does not provide the Synchronizer with unchanged records. Moreover, unlike a regular Translator it provides deleted records,  
15 which the regular Translators does not.

In order for such databases to be synchronized without resorting to treating them as regular databases, the Synchronizer transforms Fast Synchronization records from the Translator into the equivalent regular database records.  
20 These transformed records are then used by the Synchronizer in the synchronization. There are two transformations which are necessary. First, the Synchronizer needs to transform deleted records received from the Fast Synchronization Translator into a regular database deletions. Second,  
25 synchronization needs to transform lack of output by the Fast Synchronization Translator into unchanged records.

The invention performs these transformations by using the History File. During the first synchronization, all records in the Fast Synchronization database are loaded  
30 into the history file. As changes, additions, and deletions are made to the Fast Synchronization database, during each of the subsequent synchronizations the same change, additions, and deletions are made to the History File.

Therefore, the History File at the end of each subsequent synchronization is an exact copy of the Fast Synchronization database.

When a Fast Synchronization Translator supplies no  
5 input for a unique ID H\_Record, the Synchronizer finds the corresponding H\_Record in the Workspace and copies it into the Workspace as a record supplied as if it were loaded by the Fast Synchronization translator itself.

Referring to Fig. 30, steps 1050-1051, the  
10 Synchronizer first verifies that there is an appropriate History File. Because the Fast Synchronizing process relies heavily on the History File, it is important to ensure that the same history file as the last Synchronization is used. Moreover, the History File is the background against which  
15 the transformation of the Translator outputs into regular Translator outputs takes place. The History File keeps a date and time stamp of the last synchronization. Each of the Fast Synchronization database (if able to) and the Fast Synchronization Translator also stores the same date and  
20 time stamp. The time and date stamp is used because it is unlikely that another History File will have exactly the same time and date entry, for the same two databases. It also identifies when last the Fast Synchronizer database and the History File contained the same records.

At the start of an incremental synchronization, the  
25 Synchronizer and the Fast Synchronization Translator compare date and time stamps. If time and date stamp synchronization parameters have changed since the previous synchronization, then the synchronization proceeds from  
30 scratch (step 1052). In a synchronization from scratch all records in the Fast Synchronization database are loaded into the History File.

SECRET 0642780

In the preferred embodiment, all records supplied as Fast Synchronization inputs have a special hidden field called Delta, which carries a single-letter value - 'D' for Delete or 'A' for Add and 'C' for Change. Records are loaded by the Fast Synchronization Translator into the Workspace (step 1054). If necessary the records are mapped when loaded. Records which are marked as changes or additions are sanitized by the Translator for the other database, but deleted records are not because their field values are going to be deleted (step 1055). Orientation analysis (Fig. 11) is performed on the records so that all deletions and changes to Fast Synchronization database records are joined with their History File counterparts in unique ID bearing CIGs (step 1107).

All History File records and their CIGs are now examined. If there is no corresponding record from the Fast Synchronization database, it means that the record was unchanged. A clone of the record is made, labelled as being from Fast Synchronization database, and joined to the H\_Record's CIG. At this point the deleted Fast Synchronization database records marked as deletions are removed from CIGs (step 1109). The Fast Synchronization records marked as changed are joined in doubleton CIGs. Those marked as additions are singletons. At this point, the synchronization can proceed as if record of a unique ID bearing regular database were just loaded into the Workspace.

Whenever we are loading from a Fast Synchronization database, all records are loaded so that at the end of synchronization the history file will be the same as the Fast Synchronization Database. Therefore, referring to <sup>Figs.</sup> ~~Fig.~~ 31A and 31B, in order to perform date range limited synchronization, the invention marks the records which fall outside the

08752490-11396  
96ETT-06425280

current and the previous date ranges. For a record marked as an addition, or during synchronizing from scratch, if the record falls outside the current date range, it is marked as Out\_Of\_Range (steps 1101 and 1153-1154). This record will  
5 be written into the History File but not into the other database or take part in the synchronization. When the Fast Synchronization database records are loaded from the History File, if they fall outside of the previous date range, they are marked as Bystander (steps 1152-1157). If a Bystander  
10 record forms a CIG with a Fast Synchronization record marked as a deletion or a change, the Bystander is marked with a Garbage flag because its field values serve no useful purpose any more: the record marked as DELETION should be deleted and the record marked as CHANGED should replace the  
15 Bystander H\_Record (step 1162).

H\_Records for which there are no inputs are transformed in the same manner as before (steps 1164-1165). If a Bystander record falls within the current date range, it is equivalent to a regular database record coming into  
20 the current date range. Therefore, the H\_Record is cloned and marked as a Fast Synchronizer record while the Bystander record is marked as Garbage (steps 1166-1171). Therefore, just like a new record of a regular database, it has no H\_Record counterpart.

25 If the user selects to abort a synchronization or selects the option to ignore a conflict or conflicts in general, some of the records loaded from the Fast Synchronization database will not be accepted and recorded in the History File. Therefore, the Translator should  
30 provide that record again at the next synchronization. However, because Fast Synchronization Translators supply only records which have been changed, deleted, or added since the previous synchronization, the records which were



not accepted will not be supplied. Therefore, in the invention, Fast Synchronization Translator waits for an acknowledgement from the Synchronizer that the record has been accepted.

5           In case no such acknowledgement is received for a record, the Translator needs to be able to provide that record again to the Synchronizer. If the database allows resetting individual Dirty bits, the Translator merely does not set that bit. If not, the Translator keeps a separate  
10 file in which it keeps a record of which Fast Synchronization records were not accepted. The file may contain the unique IDs of those records. The Translator then uses that file to provide the synchronizer with those records during the next synchronization.

15           Other embodiments are within the following claims.